170-TP-008-001

Writing HDF-EOS Point Products for Optimum Subsetting Services

Technical Paper

Technical Paper--Not intended for formal review or government approval.

December 1996

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Shaun de Witt /s/	12/13/96
Shaun de Witt, Senior Software Engineer EOSDIS Core System Project	Date

SUBMITTED BY

Steve Marley /s/	12/13/96
Steve Marley, Senior Software Architect, EOSDIS Core System Project	Date

Hughes Information Technology Systems
Upper Marlboro, Maryland

This page intentionally left blank.

Abstract

This document provides a guideline for writing point data into HDF-EOS files to allow for optimum subsetting performance. HDF-EOS builds on the standard HDF libraries produced by NCSA for storing data, and extends it to support commonly used data structures used for earth science data. It allows for the construction of grids (in standard projections), swaths and point data sets. HDF and HDF-EOS are self-describing data formats and allow a great deal of flexibility in the internal organization of the data. Requirements exist within ECS to allow for subsetting and subsampling services to performed on these data sets; specifically subsetting by geographic co-ordinate, altitude and time. Since there are a large number of methods for organizing the data, much specific code would need to be written for each product to allow these services to take place and this variability can lead to difficulty in users interpreting data from different providers. This document is intended as a guide to writing point data in a form suitable to allow subsetting as described above, minimizing the amount of specialist code which would be needed for these services to take place, and provide a loose standard to allow other users to interpret the data readily.

Keywords: Point, HDF, HDF-EOS, format, subsetting, services.

This page intentionally left blank.

Contents

Figures

Abbreviations and Acronyms

1. Introduction

1.1 Purpose

Given the self-describing nature of HDF-EOS, there are a vast number of ways in which data could be organized within a file. Having many different data organization approaches across the EOS data sets will lead to inefficiencies in developing common data type services, like subsetting, that are sensitive to data organization. In order to facilitate the development of subsetting services, this document provides a set of guidelines for writing HDF-EOS formatted grid files which will require these services.

The author wishes to thanks the HDF-EOS development team, especially Doug Ilg and Joel Gales, for their input to this document.

1.2 Organization

This paper is organized as follows:

Section 2 gives a brief overview of the subsetting which will be performed by ECS on request. Later sections deal specifically with the data organization for a number of cases. These take the user step by step through creating a point file, together with simple code examples.

1.3 Reference Documents

The following documents were used in the preparation of this paper.

HDF-EOS User's Guide, 170-TP-005-001, 6/96

Science Data Processing Segment (SDPS) Database Design and Database Schema Specifications for the ECS Project, 311-CD-002-004, 12/95

Release-B SDPS Database Design and Database Schema Specifications, 311-CD-008-001, 5/96

1.4 Applicable Documents

The following material is related to this document and may be useful further reading.

Release A SCF Toolkit Users Guide, 333-CD-003-004

SDP Toolkit Primer for the ECS Project, 194-815-SI4-001, 4/95

HDF User's Guide, Version 4.0r2, NCSA, University of Illinois at Urbana-Champaign, 7/96

Writing HDF-EOS Grid Products for Optimum Subsetting Services, 170-TP-007-001, 12/95

Writing HDF-EOS Swath Products for Optimum Subsetting Services, 170-TP-009-001, 12/95

Questions regarding technical information contained within this Paper should be addressed to the following ECS contacts:

- ECS Contacts
  - Doug Ilg, Senior Software Engineer, (301) 925 0780, dilg@eos.hitc.com (HDF-EOS related questions)
  - Joel Gales, Senior Software Engineer, (301) 925 0782, jgales@eos.hitc.com (HDF-EOS related questions)
  - Kate Senehi, Senior Designer, (301) 925 4035, ksenehi@eos.hitc.com (Service related questions)
  - Alward Siyyid, Senior Software Engineer, (301) 925 0579, asiyyid@eos.hitc.com (General questions)

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Systems
1616 McCormick Drive
Upper Marlboro, Maryland 20774-5372

2. Services Provided on Point Data by ECS

In the Release B timeframe ECS will provide services for specific data sets as requested by Instrument Teams in agreements between themselves and ECS. All products requiring some archiving, either permanently or with a limited lifetime, will have the standard set of services, insert, update, and acquire. In addition, higher level services will be available for specific ESDT's. (one data set equates to one ESDT). Amongst these services are subset and subsample.

It should be noted that subsetting and subsampling will only be performed on HDF-EOS format files during the release B timeframe, since it is intended to utilize functionality within HDF-EOS to perform most of the services. These high level services will only be provided if Instrument Teams have indicated a need to perform them on particular ESDT's. For point data ECS does not intend to support any default subsampling of data and will only provide default subsetting services by latitude, longitude, altitude/depth, time and parameter. ECS will not by default support these higher level services on other dimensioned data.

The information contained in this and subsequent chapters is only relevant for data in HDF-EOS Point format. Swath and Grid data structures are dealt with separately in the technical papers 170-TP-009-001 and 170-TP-007-001. While this document does only provide guidelines to writing Point files requiring subsetting, it is strongly recommended that data providers follow the instructions contained for all files. This would allow users unfamiliar with a product to quickly and easily understand the contents. In addition, if a product requiring subsetting does not conform to these guidelines, the Data Provider must provide ECS with very detailed format information regarding the file organization, grid, field and dimension naming conventions, etc.

2.1 Choosing an HDF-EOS Structure

HDF-EOS has been developed as a part of the EOSDIS Core System, with the intent that it provide data producers with a standard format for earth science data archived within ECS. It is built on the widely used HDF libraries produced by NCSA, and is fully compatible with HDF. What it does provide is support for commonly used structures in earth science and remote sensing applications, namely a point, swath and grid structure. It is important for a data provider to decide which, if any, of these structures should be used for a particular implementation. The decision tree in Figure 2.1 may help in this choice, but is intended for guidance only.

Note for the box reading "Use either HDF-EOS point or grid structure" the choice is up to the user. Using a grid will make the file larger, and many fill values may exist but the code is simpler. In addition HDF compression utilities may help reduce the file size. Use of the point structure may require more in terms of coding, but will result in a smaller file.

The first decision box requires the reader to understand what is meant by spatially related. This could be interpreted in a number of ways, but it is meant to indicate that the data is spatially contiguous. As an example, an image of an area would be spatially related (all of the "pixels" are connected directly to others), while a set of meteorological reporting stations in different cities would not be related.

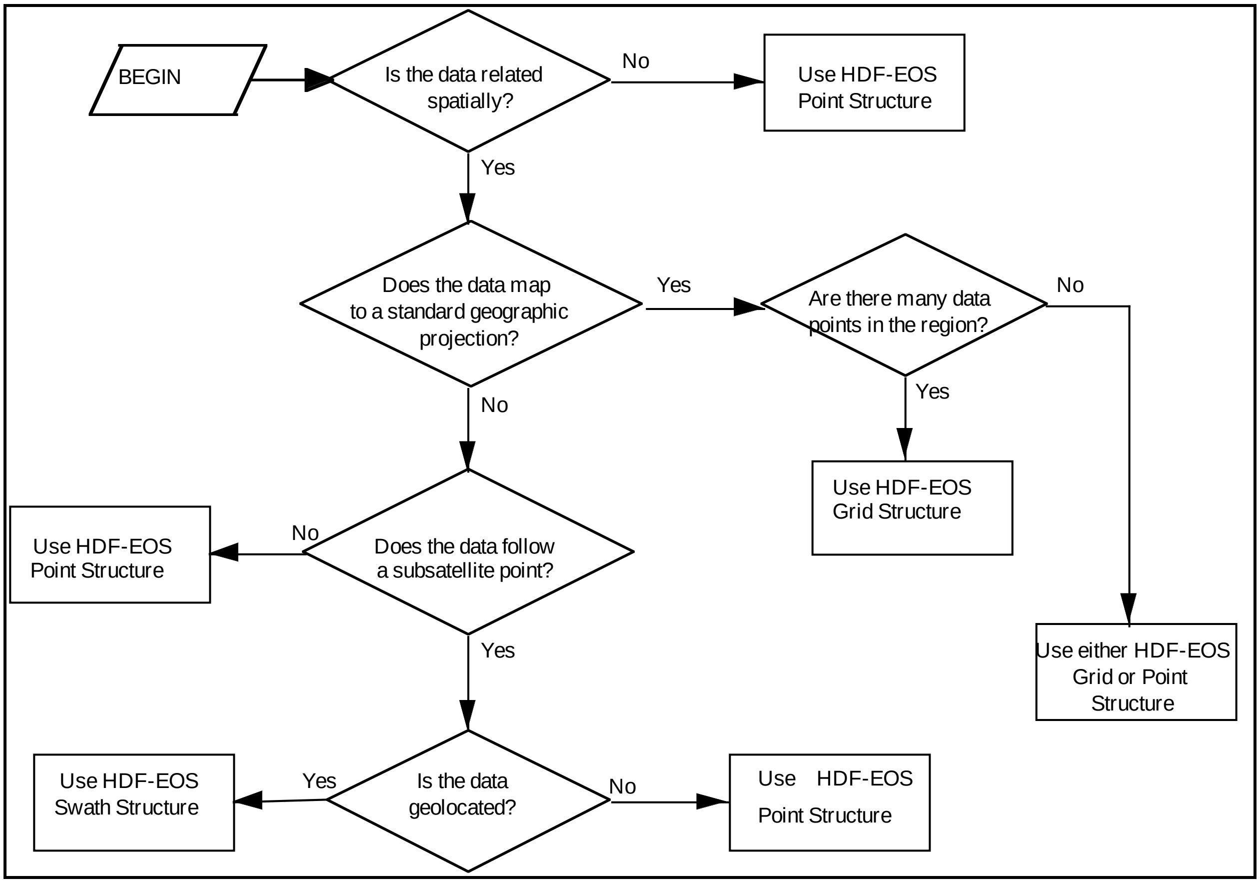

Figure 2-1. Decision Tree for Choosing an Appropriate HDF-EOS Structure

The remainder of this document only deals with the point structure.

2.2 Understanding the HDF-EOS Point Structure

Before continuing it is worthwhile to give some background on the point structure since it differs significantly from grid and swath structures. This section should be read in conjunction with the Point Structure Information in the HDF-EOS Users Guide.

Point data is the most general structure available in HDF-EOS. In essence the data are stored in a similar manner to a relational database. A point object can consist of one or more "levels". Each "child" level is linked to its "parent" via a common field (the key as an example, consider the case of data from five fixed buoys at different times, each measuring significant wave height. This could be implemented as the following two linked tables.

Table1

Lat	Long	Buoy ID
25.2645	-91.2564	0126
22.3549	-93.4657	3564
23.2564	-89.2546	1256

Table2

Buoy ID	Time	Wave Height	Temp(C)
0126	01:26	2.54	18.4
0126	05:56	3.58	18.2
3564	06:28	12.64	16.4
1256	08:12	7.58	17.1
1256	09:58	7.76	17.2
0126	09:59	4.23	20.1
3564	10:16	10.23	17.5

Figure 2-2. Table Representation of Point Data Structure

In this diagram, there are two tables; one containing fixed information regarding the buoys and one containing time ordered data values. These two tables are related via the buoy identifier code. In terms of the HDF-EOS point structure this would be a single point structure with two levels, the lowest (zeroth) level would contain the static information while the next (level = 1) would contain the more dynamic information, including the buoy ID. Note that this could equally well be represented as three separate point structures, one for each row in table 1, and the first level would only contain the data relevant to that buoy.

Even a single level point structure can get quite complex. Take the case of a single location that collects temperature data every 15 minutes for a day starting at midnight, in addition to hourly mean temperatures and accumulated rainfall. This could be implemented as a single level point which would notionally look like that below (Figure 2-4). The only difference would be that there would only be one row, and the values for the geophysical parameters would be stored as arrays within the cell.

Lat	Long	StartTime	Temp	MeanTemp	Accum. Rainfall
x	y	00:00	12.0	12.0	1
			12.2	12.2	3
			11.7	12.2	4
			12.1	12.5	6
			12.5	12.4	6
			12.4	12.6	7
			:	:	:

Figure 2-3. Example of Using Arrays For Data Storage

This sort of point structure can be placed into a single level by virtue of the fact that users can specify the "order", i.e. the number of array elements, for a column in the point table.

Further examples of single level and multi-level point structures are discussed in the HDF-EOS Users Guide.

2.3 Subsetting Service

Subsetting of a file is a means of extracting a portion of a data set relevant to a user's application. It is more sophisticated than subsampling. Subsetting can be used if a user is only interested in a particular region, parameter or altitude for example. There are several types of subsetting which are available for HDF-EOS point files, and these are dealt with in the following subsections.

2.3.1 Subsetting by Region

Subsetting by region allows you to specify a geographic region of interest. For example if the data set contains information over the whole world, but the user is only interested in data from the amazonian basin, then the user could select only that geographic region by passing in information regarding the bounding coordinates. These will represent two opposite corners of the region to be subsetted in decimal degrees. Note that only a bounding rectangle of data may be subsetted; more complex shapes are not supported by HDF-EOS.

2.3.2 Subsetting by Altitude

Subsetting by altitude can allow a user to specify the range of altitudes in which they are interested. For example, if a data set contained a vertical profile of wind speed between 0 and 3000 m above sea level, but a user is only interested in windspeed between 500 and 1000m, these may be passed in as arguments to the vertical subsetting routines. Note that the units is important here, since the vertical profile may be given in one of a number of units (for example, meters, hPa, isentropic levels, etc.). Since often there is no simple means of providing conversion between units this will not be performed by the subsetting routines. It is the responsibility of the user to ensure that the correct units are supplied when requesting subsetting.

2.3.3 Subsetting by Time

Along the same lines as subsetting by altitude, subsetting by time can allow a user to select only data from the temporal range. The method of subsetting is similar to that for altitude, where the user must supply a start and stop time to extract the required data. For temporal subsetting some unit conversion is possible between the time format requested and the time format in which the data is archived.

2.3.4 Subsetting by Parameter

In addition to the above, it will also be possible to subset by parameter. This parameter may be either geophysical (e.g. temperature) or product specific (e.g. rotation speed). Subsetting on more than one parameter will be permitted by allowing users to specify an array of parameters. The only restriction is that the parameter names requested must match those given in either of the collection level metadata attributes ParameterName or ECSVariableKeyword.

2.4 Output File Organization

Whenever a request for subsetting or subsampling has been successfully completed, a new HDF-EOS file will be generated. Whenever possible, the organization of the data will be similar to that in the original file, with the same object names and field names. The inventory and archive metadata will be copied from the original file to the subsetted/subsampled file, and the core attributes relating to bounding coordinates and time will be updated if necessary. HDF-EOS will generate new structural metadata. However, product specific and other core attributes will not be altered. Where an object (grid, point or swath) contains no relevant data, that object will be completely omitted from the subsetted/subsampled file. Similarly, if a field contains no relevant data, it will be completely omitted. No "place holders" will be left showing where original data was located.

This page intentionally left blank.

3. Writing a File Containing Simple Point Data

This section discusses the use of simple point data with only a single level to each point. This is by far the simplest form of point structure. While most point data will be more complex than this, this section does provide a useful starting point for developing more complex structures.

This scenario could be used when the data consists of a snapshot in time from a number of points. The preferred method of writing this form of data is shown below. In this case, the point structure(s) within the file must be all at a single level.

Step 0: Open the file using PTopen.

Step 1: Create a point structure using PTcreate.

Step 2: Define a level for the point structure using PTdeflevel. The field list for this level should include the values "Latitude" or "Colatitude" and "Longitude" if regional subsetting is required. If altitude subsetting is required then one of the fields in the field list must be "altitude_*units*", where *units* is defined below. If temporal subsetting is required then the field list must include the name "Time". Note that the names in quotes are case sensitive and must be given exactly as specified, although the order is not important.

Step 3: Detach from the point object using PTdetach. This is needed to fix the point structure.

Step 4: Reattach to the point object using PTattach.

Step 5: Write data to the level just defined using PTwritelevel.

Step 6: Detach from the point structure using PTdetach.

Step 7: Repeat steps 1 to 6 for each point structure required.

Step 8: Close the file using PTclose.

3.1 Restrictions to Units

There are some restrictions to the units which altitude information must be supplied (Note that altitude and depth are used synonymously throughout this report). These restrictions are needed to limit the ways in which altitude values may be presented. One of the restrictions is that the field containing dimension values must contain either integer or real values; no string values are allowed. This is a limitation imposed by HDF-EOS.

In the case of subsetting by altitude (or depth; the two terms are used synonymously), the units may be one of the following:

Pa, hPa, kPa, atm, meters, km, fathoms, millibars, theta, sigma, Kelvin, Celsius.

Some unit conversion will be performed during a subsetting request. Conversion will be performed between the following units:

meters and km

Pa, hPa, kPa, millibars and atm

Kelvin and Celsius

No other conversions will be performed within Science Data Server due to the often complex relationship between them (for instance Pa will not be converted to meters). It may be that sophisticated Clients can perform additional conversions prior to subsetting.

If a time field exists, and it is required to subset the data by time, then it is strongly recommended that the time be given in TAI93 (the number of continuous seconds since 12:00:00 on 1. Jan. 93). However, any time units could be used. If a different time format is to be used, then the data provider must supply ECS with the information about the chosen format to allow temporal subsetting to take place generically.

3.2 Sample Code for Writing a Simple Point Product

Since the simple point structure can be used to store a number of types of data, two examples will be looked at in this section. One is a snapshot of data from a number of different locations, while one is data from a single location at different times.

3.2.1 Snapshot at Different Locations

In this example, it is assumed that the product contains data from 10 different receiving stations (Atlanta, Washington, Los Angeles, Phoenix, Chicago, New York, Seattle, Denver, Dallas and Miami). It is assumed that the data are measurements of windspeed, temperature and dew point taken at a single time. In this case, the data may be considered to look like the table below. Since the time is inherently known (from the metadata within the HDF-EOS file) there is no need to include a time field within the point structure.

Table 3-1. Input to Sample Code

Latitude	Longitude	WindSpeed	Temperature	DewPoint
33.75	-84.38	15.5	29.6	20.1
38.92	-77.00	5.0	26.8	20.5
34.07	-118.25	12.6	19.5	5.0
33.3	-112.03	0.0	32.1	6.2
41.83	-87.75	12.2	16.5	10.0
40.72	-74.00	8.9	17.2	12.2
47.35	-122.20	13.6	14.3	14.3
39.44	-104.98	5.1	21.6	6.5
32.78	-96.82	3.9	26.8	9.8
25.77	-80.20	7.4	28.2	25.0

In the code below, it is assumed that the data are read from an ancillary ASCII file which contains above rows(without the column headings). It is also assumed that the user wishes to store all of the data in a single point. The data could equally well be stored as 10 points; one for each location. The former method is preferred since this lends itself to subsetting by area much more conveniently and reduces the file size. This example would allow subsetting by area and parameter only since there is no altitude or time component. The code sample below is written in C and omits error checking for clarity.

```
#include <stdio.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>

#define STATIONS 10

int main()
{
int32 ptID;
int32 fileID;
int32 fieldType[5];
int32 fieldOrder[5];
intn  hdfRtn;
int   rtn, n;
```

```
char  fieldList[255];
char  dataBuf[10000];
char  *dataBufpntr;
double lat, lng;
float  wind, temp, hum;
FILE* fp;

/* Open the point file.  It is assumed that the file does not already exist */
fileID = PTopen("PointExample.dat", DFACC_CREATE);

/* Create a point object within the file */
ptID = PTcreate(fileID, "Met Reports");

/* Define the field list.  This will be used as input to PTdeflevel */
sprintf(fieldList, "Latitude, Longitude, WindSpeed, Temperature,
RelHumidity");

/* Define the types for each of the fields listed above */
fieldType[0] = DFNT_FLOAT64;  /* Latitude field */
fieldType[1] = DFNT_FLOAT64;  /* Longitude field */
fieldType[2] = DFNT_FLOAT32;  /* WindSpeed field */
fieldType[3] = DFNT_FLOAT32;  /* Temperature field */
fieldType[4] = DFNT_FLOAT32;  /* Humidity field */

/*
Define the order for each field.  This order is actually the array size for
the field list variables, zero based.  Since in this instance, all are single
valued, the order for each is zero.
*/
fieldOrder[0] = 0;
fieldOrder[1] = 0;
fieldOrder[2] = 0;
fieldOrder[3] = 0;
fieldOrder[4] = 0;

/* Now we can define the level for this structure */
```

```
hdfRtn = PTdeflevel(ptID, "Station Values", fieldList, fieldType, fieldOrder);


/*
Now to fix the point structure it is necessary to detach and reattach to the
point object.
*/
hdfRtn = PTdetach(ptID);
ptID = PTattach(fileID, "Met Reports");


/* Open the ASCII file containing the values to be inserted */
fp = fopen("RawValues.dat", "r");
n = 0
dataBufpntr = dataBuf
while (fscanf(fp, "%lf %lf %f %f %f", &lat, &lng, &wind, &temp, &hum) != -1)
{
      n++;
      memcpy(dataBufpntr, &lat, sizeof(double));
      dataBufPntr += sizeof(double);
      memcpy(dataBufpntr, &lng, sizeof(double));
      dataBufPntr += sizeof(double);
      memcpy(dataBufpntr, &wind, sizeof(float));
      dataBufPntr += sizeof(float);
      memcpy(dataBufpntr, &temp, sizeof(float));
      dataBufPntr += sizeof(float);
      memcpy(dataBufpntr, &hum, sizeof(float));
      dataBufPntr += sizeof(float);
}
fclose (fp);


/* We can now write this data to the point object */
hdfRtn = PTwritelevel(ptID, 0, n, dataBuf);


/* Finally detach and close the file */
hdfRtn = PTdetach(ptID);
hdfRtn = PTclose(fileID);
```

```
}
```

3.2.2 Snapshot at Different Times

This example assumes that all of the data in the point structure comes from a single known location and represents a series of temporal values. It is assumed that the measurements represent temperature, wind speed and accumulated rainfall taken four times an hour covering one day. The time units are assumed to be in TAI93. No geolocation information is included because it is assumed the data is taken from a known location which will be contained within the metadata. If geolocation information were added, then it would be more appropriate to use a two level point structure. Again, it is assumed that initially the data are contained in a simple ASCII file, with one line for each hour's worth of data, and four columns separated by spaces. This type of example lends itself to subsetting by time and parameter only since there are no geolocation or altitude data associated.

```
#include <stdio.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>

int main()
{
int32 ptID;
int32 fileID;
int32 fieldType[4];
int32 fieldOrder[4];
intn  hdfRtn;
int   rtn, n;
char  fieldList[255];
char  dataBuf[10000];
char  *dataBufpntr;
double time;
float  wind, temp, rain[4];
FILE* fp;

/* Open the point file.  It is assumed that the file does not already exist */
fileID = PTopen("PointExample.dat", DFACC_CREATE);
```

```
/* Create a point object within the file */
ptID = PTcreate(fileID, "Chicago");

/* Define the field list.  This will be used as input to PTdeflevel */
sprintf(fieldList, "Time, WindSpeed, Temperature, AccumulatedRainfall");

/* Define the types for each of the fields listed above */
fieldType[0] = DFNT_FLOAT64;  /* Time field */
fieldType[1] = DFNT_FLOAT32;  /* WindSpeed field */
fieldType[2] = DFNT_FLOAT32;  /* Temperature field */
fieldType[3] = DFNT_INT32;    /* Rainfall field */

/*
Define the order for each field.  This order is actually the array size for
the field list variables, zero based.  Since in this instance the first three
fields are single values, there order is zero, but the last has 4 values, its
order is 3
*/
fieldOrder[0] = 0;
fieldOrder[1] = 0;
fieldOrder[2] = 0;
fieldOrder[3] = 3;

/* Now we can define the level for this structure */
hdfRtn = PTdeflevel(ptID, "DailyData", fieldList, fieldType, fieldOrder);

/*
Now to fix the point structure it is necessary to detach and reattach to the
point object.
*/
hdfRtn = PTdetach(ptID);
ptID = PTattach(fileID, "Chicago");

/* Open the ASCII file containing the values to be inserted */
fp = fopen("RawValues.dat", "r");
n = 0
dataBufpntr = dataBuf
```

```
while (fscanf(fp, "%lf %f %f %d %d %d %d", &time, &wind, &temp, &rain[0],
&rain[1], &rain[2], &rain[3]) != -1)
{
      n++;
      memcpy(dataBufpntr, &time, sizeof(double));
      dataBufPntr += sizeof(double);
      memcpy(dataBufpntr, &wind, sizeof(float));
      dataBufPntr += sizeof(float);
      memcpy(dataBufpntr, &temp, sizeof(float));
      dataBufPntr += sizeof(float);
      memcpy(dataBufpntr, &rain, 4*sizeof(float));
      dataBufPntr += 4*sizeof(float);
}
fclose (fp);

/* We can now write this data to the point object */
hdfRtn = PTwritelevel(ptID, 0, n, dataBuf);

/* Finally detach and close the file */
hdfRtn = PTdetach(ptID);
hdfRtn = PTclose(fileID);
}
```

4. Writing Point Structures Containing Multiple Levels

Most point products will not map easily or sensibly to the simple point structure defined in section 3. Additional levels may be needed for example if the data consists of a time series from different locations or from non-fixed points (e.g. shipboard measurements). There can be any number of levels defining a point structure, but for ease of use and understanding, it is recommended that only three levels be used. An example of a three level point structure could be balloon data from a number of experiments launched at the same time but with different data retrieval rates (Figure 4-1).

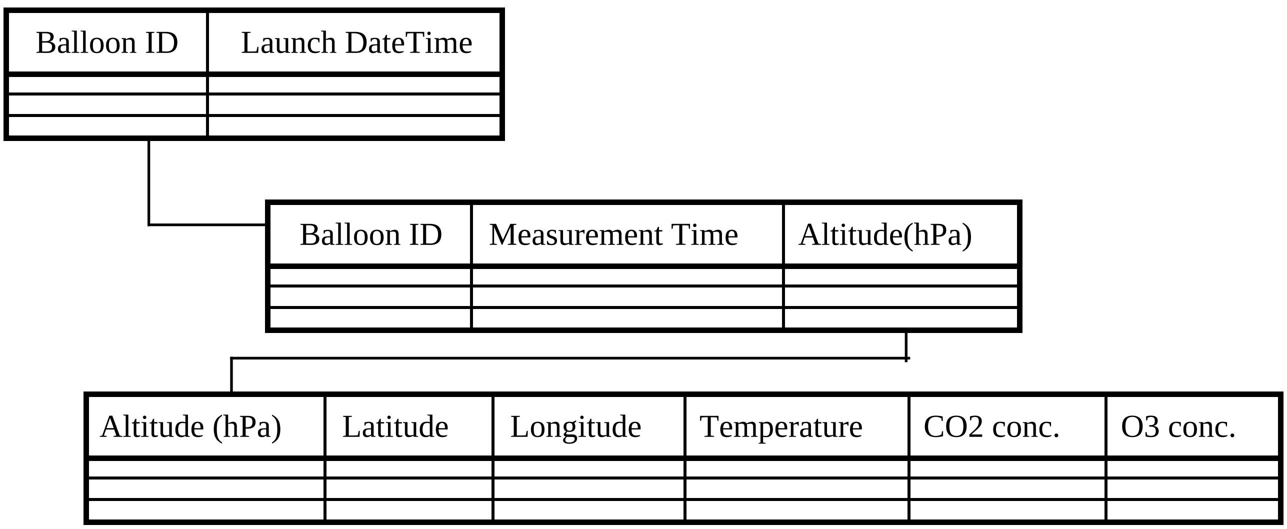

Figure 4-1. An Example of a Three Level Point Structure

The decision on how many layers to implement is made by the data provider. There will clearly be some trade off between complexity in creating the data and understanding what users want from the data. For example, in the above figure, it has been assumed that users will be interested in obtaining the data and comparing positions from the same altitudinal layer, but that the actual time is relatively unimportant. If, however, the time information is important, then this structure could be leveled to only two layers; the zeroth level being as shown and combining the information in the second third levels.

4.1 Creating Point Structures with more than One Level

As implied above, there is a certain amount of effort needed to organize a point file to make the data both easily understood and easily produced. However, once the number of levels is decided upon, then there is a common approach recommended for creating point structures. This is shown below.

Step 0 : Open the grid file using PTopen.

Step 1 : Create a point structure using PTcreate. The structure has no special significance.

Step 2 : For each level required in the structure, define the field list, order and types. This process must be done without HDF-EOS calls, but is used as input to later calls. **Note**: When defining the field lists for each level, there must be a repeated field name between parent and child levels. Having defined this information, use PTdeflevel to define each level. **Note**: Depending on the forms of subsetting required, the following fields must be defined:

Regional Subsetting: Field names of "Latitude" (or "Colatitude") and "Longitude" must be defined.

Temporal Subsetting : The field name "Time" must be defined. **Note**: This should preferably be given in TAI93 (see Section 3.1.1)

Altitude Subsetting : A field name of the form "altitude_*units*" must be defined (see Section 3.1.1 for list of valid units).

Parameter Subsetting : It is strongly recommended that data field names within the point structure are given the same name as one of the collection level metadata attributes ParameterName or ECSVariable (although this is not mandatory, if this is not followed the data provider must provide ECS with the mapping of field name to parameter).

Step 3 : Define the linkage between parent and child levels using PTdeflinkage.

Step 4 : Detach from the point structure using PTdetach. This is to allow the linkages and fields to be set within the structure.

Step 5 : Reattach to the point structure using PTattach.

Step 6 : Write the appropriate data to each level using PTwritelevel.

Step 7 : Detach from the grid using PTdetach.

Step 8: Repeat steps 1 through 7 for each required point structure.

Step 10 : Close the file using PTclose.

4.2 Sample Code for Writing Multi-Level Point Structures

The following code sample assumes that the user is trying to write data into the point structure shown in Figure 4-1. This ample is written in C and error checking has been omitted. Also the input of the data values has been omitted, although comments indicate where data is assumed to already be available.

```
#include <stdio.h>
#include <hdf.h>
#include <mfhdf.h>
#include <HdfEosDef.h>

int main()
{
int32  ptID;
int32  fileID;
int32  launchtime;
int32  fieldOrder0[2], fieldType0[2];
int32  fieldOrder1[3], fieldType1[3];
int32  fieldOrder2[6], fieldType2[6];
char   string0[255];
char   string1[255];
char   string2[255];
char   databuf[10000];
char   ballonID[8];     /* Each balloon has 8 character identifier */
char   *pntr;


/* Open the point file */
fileID = PTopen("mypointfile.hdf", DFACC_CREATE);

/* Create the point object */
ptID = PTcreate(fileID, "Balloon Data");

/* Define the field list for each level. In this case we have 3 levels */
sprintf(string0, "BalloonID, LaunchDateTime");
```

```
fieldType0[0] = DFNT_CHAR8;
fieldType0[1] = DFNT_INT32;
fieldOrder0[0] = 0;
fieldOrder0[1] = 0;
hdfRtn = PTdeflevel(ptID, "Launch Info", string0, fieldType0, fieldOrder0);

sprintf(string1, "BalloonID, Time, altitude_hPa");
fieldType1[0] = DFNT_CHAR8;
fieldType1[1] = DFNT_INT32;
fieldType1[2] = DFNT_INT16;
fieldOrder1[0] = 0;
fieldOrder1[1] = 0;
fieldOrder1[2] = 0;
hdfRtn = PTdeflevel(ptID, "Observation Info", string1, fieldType1,
fieldOrder1);
hdfRtn = PTdeflinkage(ptID, "Launch Info", "Observation Info", "BalloonID");

sprintf(string2, "altitude_hPa, Latitude, Longitude, Temperature, CO2, O3");
fieldType2[0] = DFNT_CHAR8;
fieldType2[1] = DFNT_FLOAT64;
fieldType2[2] = DFNT_FLOAT64;
fieldType2[3] = DFNT_FLOAT32;
fieldType2[4] = DFNT_FLOAT32;
fieldType2[5] = DFNT_FLOAT32;
fieldOrder2[0] = 0;
fieldOrder2[1] = 0;
fieldOrder2[2] = 0;
fieldOrder2[3] = 0;
fieldOrder2[4] = 0;
fieldOrder2[5] = 0;
hdfRtn = PTdeflevel(ptID, "Data", string2, fieldType2, fieldOrder2);
hdfRtn = PTdeflinkage(ptID, "Observation Info", "Data", " altitude_hPa ");

/* Now detach and reattach to fix the overall structure and linkage */
hdfRtn = PTdetach(ptID);
```

```
ptID = PTattach("Balloon Data");


/*

 From this point onwards it is assumed that databuf has been filled with
values for the appropriate level.  For an example of how to achieve this, see
the code examples in section 3.  For clarity it is assumed that the zeroth
level contains 6 rows (one for each mission), the first level contains 50
levels (a total of 50 observations from all of the missions) and the last
level also contains 50 elements.  Clearly there is a case for combining the
first and second levels in this instance.

*/

hdfRtn = PTwritelevel(ptID, 0, 6, databuf);

hdfRtn = PTwritelevel(ptID, 1, 50, databuf);

hdfRtn = PTwritelevel(ptID, 2, 50, databuf);


/* Finally detach from the point structure and close the file */

hdfRtn = PTdetach(ptID);

hdfRtn = PTclose (fileID);

exit(0);

}
```

This page intentionally left blank.

Abbreviations and Acronyms

atm	atmospheres (unit of pressure measurement)
ECS	EOSDIS Core System
ESDT	Earth Science Data Type
HDF	Hierachical Data Format
hPa	hectoPascal (unit of pressure measurement)
JD	Julian Date
kPa	kiloPascal (unit of pressure measurement)
MJD	Modified Julian Date
Pa	Pascal (unit of pressure measurement)
sbt	satellite binary time
UTC	Universal Time Coordinates